Patent
Docket No. 10011596-1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

APPEAL NO. __________________

In re Application of:
Christophe de Dinechin et al.

Confirmation No. 5117

Serial No. 09/873,875
Filed: June 4, 2001

Examiner: Lillian Vo
Art Unit: 2195

For: **CONTEXT-CORRUPTING CONTEXT SWITCHING**

APPEAL BRIEF

Hugh P. Gortler, Esq.

Hewlett-Packard Company
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

(949) 454-0898

INDEX

1. REAL PARTY IN INTEREST

The real party in interest is the assignee, Hewlett-Packard Development Company.

2. RELATED APPEALS AND INTERFERENCES

No appeals or interferences are known to have a bearing on the Board's decision in the pending appeal.

3. STATUS OF CLAIMS

Claims 1-26 are pending.

Claims 1-26 are rejected.

The rejections of claims 1-26 are being appealed.

4. STATUS OF AMENDMENTS

No amendment was filed subsequent to latest office action dated Dec. 11, 2006.

5. SUMMARY OF CLAIMED SUBJECT MATTER

An operating system (OS) can be run as application on a computer using a virtual machine. The OS that is run as the application is referred to as the "guest OS," and the underlying OS is referred to as the "host OS." For example, a Windows-based OS can be run as an application on top of a host Linux OS. Running the Windows-based OS as an application allows programs written for a Windows environment to be run on top of the host Linux OS.

The guest OS usually has a different “context” than the host OS. A context embodies the accessible state of the computer’s central processing unit (CPU). The context includes values of CPU registers.

The virtual machine can allow the guest OS and the host OS to run concurrently by properly restoring the context of the guest OS whenever control is transferred from the host OS to the guest OS, and by similarly restoring the context of the host OS when control is transferred from the guest OS to the host OS. This process is called “context switching.”

Certain CPU architectures, such as IA-64, have prohibitions against, and difficulties with, completely saving and restoring the entire context of a guest OS or a host OS. Specifically, certain registers containing context might be corrupted during context switching. If the entire context cannot be saved, the guest OS or host OS might behave incorrectly and crash.

A simple example in paragraph 17 of the application illustrates a problem that such architectures have with context switching. Context of a host OS is contained in registers X, I, J and K. During context switching, register X is used to store an address that indicates where the context will be saved. However, the register X is overwritten during context switching, whereby the context of register X is lost before it can be saved.

This problem is overcome by the method of claim 1, the method of claim 11, the apparatus of claim 12, and the software of claim 22.

Base claim 1

Claim 1 recites a method of switching context on a processor (element 110 in Figure 1). Referring to Figure 2 and paragraphs 20-21 of the specification, the method comprises saving the context under software control using an inconsequential register (312); and preventing the processor from changing the context while the context is being saved (310).

The inconsequential register can be used to store context, for example, by storing an address that indicates where the context will be saved. This example is described in paragraph 23 of the application.

According to paragraph 21 of the specification, an inconsequential register is a register that is not used by the host OS at a predetermined interruption point (PIP). A point can be predetermined if the context is saved under software control (which is synchronous) instead of hardware control (which is asynchronous). According to paragraph 22 of the application, since the context of a host OS is saved at the PIP, it is known which registers the host OS uses and which registers the host OS does not use. Therefore, the inconsequential register(s) at the PIP can be identified. The inconsequential registers can be corrupted by a virtual machine application without affecting the host OS. In particular, the context switching process can use the inconsequential registers as a temporary storage in lieu of the privileged registers.

Base claim 12

Reference is made to Figures 1-2 and paragraphs 20-21 of the specification. Claim 12 recites apparatus comprising a processor (110) including a plurality of registers (110) and a virtual machine application (212). The virtual machine application (212) commands the processor (110) to switch context by saving the context under software control using an inconsequential register of the processor as temporary storage (312). The virtual machine application also prevents the processor from changing the context while the context is being saved (310).

<u>Base claim 22</u>

Reference is made to Figures 1-2 and paragraphs 20-21 of the specification. Claim 22 recites software (212) comprising instructions for commanding a processor to switch context by saving the context under software control using an inconsequential register of the processor as temporary storage (312); and preventing the processor from changing the context while the context is being saved (310).

<u>Base claim 11</u>

Reference is made to Figures 1-2 and paragraphs 20-22 of the specification. Claim 11 recites a method of switching context between a host OS (210) and a virtual machine (212) on a processor (110). The processor (110) has privileged registers and access to other memory (112). Referring additionally to Figure 3 and paragraph 29, the method comprises giving the virtual machine access to the privileged registers (point A in Figure 3); using at least one privileged register as temporary storage to save the context in the other memory at a predetermined interruption point (between points A and B in Figure 3); and preventing the processor from changing the context while the context is being saved (310). The virtual machine controls the context switch.

6. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

a. Base claim 1 is rejected under 35 U.S.C §102(b) over Bitar U.S. Patent No. 5,872,963.

b. Base claims 12 and 22 are rejected under 35 U.S.C §103(a) over Bugnion U.S. Patent No. 6,496,847 in view of Bitar.

c. Base claim 11 is rejected under 35 U.S.C §103(a) over Bugnion in view of Ooi U.S. Patent No. 5,043,878.

d. Claims 1-11 and 22-26 are rejected under 35 USC §101 as being directed to non-statutory subject matter.

e. Claim 26 is objected to for not further limiting the subject matter of base claim 1.

7. <u>ARGUMENTS</u>

I
REJECTION OF BASE CLAIM 1 UNDER 35 U.S.C §102(b) OVER BITAR U.S. PATENT NO. 5,872,963

A memory register is a special, high-speed storage area within the a CPU. This fact is supported by the specification, which refers in general to "CPU registers" and to a CPU "having" registers (see paragraphs 2 and 11). The specification also states that an IA-64 processor "includes" general registers, branch registers, and other types of registers (see paragraph 30).

In contrast, system or main memory is separate from the CPU, and it is connected to the CPU by a memory bus. Paragraph 13 of the specification refers to main memory as random access memory (RAM) 112.

Bitar does not teach or suggest how memory registers are used to save context. It follows that Bitar does not teach or suggest saving context using an inconsequential register.

Bitar describes context switching at col. 8, lines 48-54. If a thread is pre-empted, the context of the thread is saved in a register save area 29 of a shared arena 22.

The register save area is part of main or system memory. Figure 2a of Bitar shows a processor 12 connected to memory 14 (col. 6, lines 41-43). The memory 14 is system memory, not register memory. The system memory 14 includes user space 20, which has a shared arena 22 per application (col. 6, lines 43-45). A shared arena 22 has a plurality of register save areas 29, where each register save area 29 contains sufficient memory locations to store the context of one user-level thread (col. 7, lines 45-48).

The office action alleges that the use of an inconsequential register is disclosed at col. 13, lines 57-65; col. 17, lines 39-41 and 63-65; and col. 18, lines 1-16. However, Bitar does not support this allegation. The passage at col. 13, lines 57-65 indicates that context is saved to register save areas 29, which is part of system memory 14. The remaining passages refer to claims 1, 3 and 4. These claims recite memory, which Bitar's detailed description describes as system memory.

The passage at col. 13, lines 57-65 does not identify a register that is inconsequential. The passage does not teach or suggest the use of a register, inconsequential or otherwise, for saving context. Moreover, Bitar does not teach or suggest preventing the processor from changing the context while the context is being saved.

Bitar does not even address the problem of a register being overwritten during context switching, whereby the context of that register is lost before it can be saved. Claim 1 recites a method that overcomes this problem. Bitar does not teach or suggest a method that overcomes context corruption during context switching.

Thus, Bitar does not teach or suggest a method having all of the features of base claim 1. Therefore, base claim 1 and its dependent claims 2-10 and 23-26 should be allowed over Bitar.

II
REJECTION OF BASE CLAIMS 12 AND 22 UNDER 35 U.S.C §103(a) OVER BUGNION U.S. PATENT NO. 6,496,847 IN VIEW OF BITAR

The office action acknowledges that Bugnion does not teach or suggest the use inconsequential registers for context switching. And, as discussed in argument I above, Bitar does not teach or suggest the use of inconsequential registers either. Therefore, the '103 rejection of base claims 12 and 22 and dependent claims 13-21 should be withdrawn.

The '103 rejection should be withdrawn for the addition reason that the holding of obviousness is flawed, both technically and legally. The office action alleges that it is obvious to modify Bugnion's system with Bitar's teachings to avoid a system crash, yet it does not support this allegation with any evidence or reasons. The office action simply makes a bald conclusion of obviousness.

Neither Bugnion nor Bitar addresses a specific problem of a register being overwritten during context switching, whereby the context of that register is lost before it can be saved (as paragraph 4 of the specification mentions, the prohibition against, and difficulties with, completely saving and restoring the entire context of a guest or host OS is present only in certain architectures). Both documents disclose storing context in system memory, yet neither document discloses how memory registers are used to store the context in the system memory.[1] Therein lies the problem.

---

[1] Col. 13, lines 30-41 of Bugnion states that any memory can be used to perform a context switch. However, the use of "any" memory won't address the problem faced by the applicant.

III
REJECTION OF BASE CLAIM 11 UNDER 35 U.S.C §103(a) OVER BUGNION IN VIEW OF OOI U.S. PATENT NO. 5,043,878

Claim 11 recites a method of switching context between a host OS and a virtual machine on a processor. The processor has privileged registers and access to other memory (e.g., main memory). The virtual machine is given access to the privileged registers, using at least one privileged register as temporary storage to save the context in the other memory at a predetermined interruption point. The processor is prevented from changing the context while the context is being saved. The virtual machine application controls the context switch.

The office action acknowledges that Bugnion does not teach or suggest the use of privileged registers (as discussed in Argument II above, Bugnion does not describe how memory registers are used to save context).

However, the office action does not acknowledge other differences between claim 11 and Bugnion. Bugnion does not teach or suggest giving access to privileged registers during a context switch and using them as temporary storage. The office action cites a passage at col. 14, lines 1-10. However, that passage merely indicates that context includes the state of privileged registers. The passage does not teach or suggest giving access to privileged registers during a context switch.

Another unacknowledged difference is that Bugnion does not teach or suggest the use of a predetermined interrupt point. According to paragraph 22 of the specification, a predetermined interrupt point is the point at which the host OS context is saved under software control (synchronous). As the office action points out, Bugnion's context switch occurs under asynchronous control (the occurrence of an interrupt – col. 17, lines 6-21).

The office action cites a passage at col. 4, lines 52-61. The passage discloses that host OS context is saved in a driver, and then context switching is performed in the driver. The passage also states that virtual machine monitor context is saved in a virtual machine monitor, and then context switching is done in the virtual machine monitor. However, the passage does not indicate how the context is saved.

The office action cites a passage at col. 4, lines 46-48. The passage states that a computer uses a set of privileged instructions. However, the passage says nothing about privileged registers being used as temporary storage to save context.

The office action cites a passage at col. 11, lines 13-15. The passage states that code is executed when an interrupt occurs.

The office action cites a passage at col. 11, lines 30-52. This passage gives an overview of Bugnion et al.'s context switch. The passage indicates that any available memory may be used to save context. The passage does not indicate whether context is saved at a predetermined interruption point, or whether privileged registers are used as temporary storage to save context.

Ooi does not teach or suggest the differences between claim 11 and Bugnion. The office action cites passages at col. 9, lines 22-30 and col. 7, lines 39-47. However, these passages are not relevant, as they relate to privileged instructions and avoiding illegal register references. These passages do not relate to context switching.

Ooi does describe context switching at col. 8, lines 32-45. Ooi mentions that context of privileged registers is saved during a context switch. However, Ooi does not teach or suggest using the privileged register as temporary storage to save context.

Moreover, Ooi is also silent about predetermined interruption points. Ooi appears to disclose hardware (asynchronous) control of context switching.

For these reasons, the ‘103 rejection of base claims 11 should be withdrawn.

IV
REJECTION OF CLAIMS 1-11 and 22-26 UNDER 35 U.S.C §101

The office action contends that the methods of claims 1-11 and 23-26 can be carried out as a mental process in conjunction with pen and paper. Claim 1 recites storing data in hardware registers. Claim 1 also recites preventing a processor from changing contexts. The office action doesn't explain how pen and paper can be used to store processor context in a hardware register, or how pen and paper can be used to prevent a processor from changing context while the context is being saved. Because claim 1 clearly recites actions that do not read on a mental process, the '101 rejections of claims 1-11 and 23-26 should be withdrawn.

The office action contends that claim 22 lacks utility and is not tangibly embodied in a manner to be executable. This contention is puzzling, since claim 22 recites "software comprising instructions for commanding a processor to switch context." Context switching is a useful operation; therefore, the software of claim 22 has utility. The software can command a processor; therefore, it has to have a tangible embodiment that can be executed. Withdrawal of the '101 rejection of claim 22 is respectfully requested.

V
OBJECTION TO CLAIM 26

Claim 26 is objected to for failing to further limit the subject matter of base claim 1. This objection was previously raised as a ‘112 rejection, so it is addressed in this appeal brief.

Claim 1 recites that an inconsequential register is used to save context, and claim 26 recites that the context itself is saved in memory other than the inconsequential register. Claim 26 is supported by an example in paragraph 23 of the specification. In that example, an inconsequential register I is used to store the address at which the context will be stored, and context in registers J, K and X is saved at that address. Thus, claim 26 further limits the subject matter of claim 1.

For the reasons above, the ‘101, ‘102 and ‘103 rejections should be reversed. The Honorable Board of Patent Appeals and Interferences is respectfully requested to reverse these rejections.

Respectfully submitted,

/Hugh Gortler #33,890/
Hugh P. Gortler, Esq.
Registration No. 33, 890

Hewlett-Packard Company
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

(949) 454-0898

Date: June 11, 2007

8. CLAIMS APPENDIX

1. (Previously presented) A method of switching context on a processor, the method comprising:

saving the context under software control using an inconsequential register; and

preventing the processor from changing the context while the context is being saved.

2. (Original) The method of claim 1, wherein the inconsequential register is used as a temporary storage in lieu of a privileged register.

3. (Original) The method of claim 1, wherein the context is saved at a predetermined interruption point.

4. (Original) The method of claim 1, wherein the context is switched between a host operating system and a virtual machine application, the virtual machine application controlling the context switch.

5. (Original) The method of claim 4, wherein the inconsequential register is used to pass information to the virtual machine application.

6. (Original) The method of claim 1, wherein the context is switched using an IA-64 processor.

7. (Previously presented) The method of claim 6, wherein the inconsequential register is a caller-save register.

8. (Previously presented) The method of claim 6, wherein the inconsequential register is a branch register.

9. (Original) The method of claim 1, further comprising restoring the context using the inconsequential register as temporary storage.

10. (Original) The method of claim 9, wherein the context is restored by using a branch register to perform an indirect branch.

11. (Previously presented) A method of switching context between a host OS and a virtual machine on a processor, the processor having privileged registers, the processor having access to other memory, the method comprising:

giving the virtual machine access to the privileged registers;

using at least one privileged register as temporary storage to save the context in the other memory at a predetermined interruption point; and

preventing the processor from changing the context while the context is being saved;

the virtual machine application controlling the context switch.

12. (Original) Apparatus comprising:

a processor including a plurality of registers; and

a virtual machine application for commanding the processor to switch context by saving the context under software control using an inconsequential register of the processor as temporary storage; and preventing the processor from changing the context while the context is being saved.

13. (Original) The apparatus of claim 12, wherein the inconsequential register is used as a temporary storage in lieu of a privileged register.

14. (Original) The apparatus of claim 12, wherein the context is saved at a predetermined interruption point.

15. (Original) The apparatus of claim 12, further comprising a host OS; wherein the context is switched between the host OS and the virtual machine application; and wherein the virtual machine application controls the context switch.

16. (Original) The apparatus of claim 15, wherein the inconsequential register is used to pass information to the virtual machine application.

17. (Original) The apparatus of claim 12, wherein the processor is an IA-64 processor.

18. (Previously presented) The apparatus of claim 17, wherein the inconsequential register is a caller-save register.

19. (Previously presented) The apparatus of claim 17, wherein the inconsequential register is a branch register.

20. (Previously presented) The apparatus of claim 12, wherein the virtual application further commands the processor to restore context using the inconsequential register as temporary storage.

21. (Original) The apparatus of claim 20, wherein the context is restored by using a branch register to perform an indirect branch.

22. (Original) Software comprising instructions for commanding a processor to switch context by saving the context under software control using an inconsequential register of the processor as temporary storage; and preventing the processor from changing the context while the context is being saved.

23. (Previously presented) The method of claim 1, wherein content of the inconsequential register is corrupted during the context switch.

24. (Previously presented) The method of claim 1, wherein using the inconsequential register includes storing an address in the inconsequential register, the address indicating a memory location at which the context will be saved.

25. (Previously presented) The method of claim 1, wherein the inconsequential register does not store context at a predetermined interruption point.

26. (Previously presented) The method of claim 1, wherein the context is stored in memory other than the inconsequential register.